



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

PLGRAM
Documentación Técnica



Presentado por Víctor Renuncio Tobarr
en Universidad de Burgos — 7 de junio de 2016
Tutor: Dr. César Ignacio García Osorio y Álgvar
Arnaiz Gonzalez

Índice general

Índice general	I
Índice de figuras	III
Apéndice A Manuales	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	9
Apéndice B Especificación de Requisitos	10
B.1. Introducción	10
B.2. Objetivos generales	10
B.3. Catálogo de requisitos	11
B.4. Especificación de requisitos	12
Apéndice C Especificación de diseño	14
C.1. Introducción	14
C.2. Diseño de datos	14
C.3. Diseño procedimental	20
C.4. Diseño arquitectónico	24
Apéndice D Documentación técnica de programación	25
D.1. Introducción	25
D.2. Estructura de directorios	25
D.3. Manual del programador	25
D.4. Compilación, instalación y ejecución del proyecto	25
D.5. Pruebas del sistema	25
Apéndice E Documentación de usuario	26
E.1. Introducción	26

ÍNDICE GENERAL

II

E.2. Requisitos de usuarios	26
E.3. Instalación	27
E.4. Manual del usuario	28

Índice de figuras

B.1. Diagrama principal de casos de uso	13
C.1. Diagrama de clases: Estructura de las gramáticas	16
C.2. Diagrama de clases: Estructura de las tablas de análisis	17
C.3. Diagrama de clases: Estructura del informe del análisis	18
C.4. Diagrama de clases: Estructura de las clases del análisis	19
C.5. Diagrama de clases: Estructura del prototipo	20
C.6. Ejemplo de una estructura de datos	21
C.7. Esquema general de las plantillas	23
E.1. Icono PLGRAM	27
E.2. Acceso a la carpeta contenedora de <i>PLGRAM</i> desde la consola	28
E.3. <i>PLGRAM</i> sin argumento gramática	29
E.4. <i>PLGRAM</i> sin argumento tipo de análisis	30
E.5. <i>PLGRAM</i> Con argumentos gramática y tipo de análisis	31
E.6. Estructura del fichero.XML	32
E.7. Importar fichero.XML en moodle	33
E.8. Lista de ficheros.XML en moodle	34
E.9. Ejemplo de cuestionario resuelto en moodle	34
E.10. Ejemplo fichero .TEX	35
E.11. Compilación de un fichero .TEX	35
E.12. Obtención del cuestionario completo	36
E.13. Obtención del cuestionario en blanco: Descomentar	36
E.14. Obtención del cuestionario en blanco	37

Apéndice A

Manuales

A.1. Introducción

En este anexo se detalla el estudio desde el punto de vista temporal y de la viabilidad del proyecto software.

La planificación es una de las tareas más importantes en el desarrollo de un proyecto software y servirá para determinar objetivos, evaluar la viabilidad del proyecto, priorizar actividades...

En la primera parte del anexo se detallará la planificación temporal del proyecto teniendo en cuenta la metodología ágil que se va a utilizar: *Scrum*. En esta fase se determinarán los elementos que forman el *Product Backlog* y la prioridad de cada uno de ellos. Debido a la metodología empleada, no se utilizará el clásico diagrama de *GANTT*. En lugar de esto, se definirá el *Product Backlog* y para el seguimiento se utilizará una herramienta de gestión especializada en metodologías ágiles, *VersionOne*, que permite el seguimiento diario de las tareas por parte del equipo de desarrollo.

En la segunda parte se justificará el desarrollo del proyecto desde diversos puntos de vista: viabilidad técnica, legal, económica, etc.

A.2. Planificación temporal

Esta sección detalla el trabajo realizado en cada *sprint* del proyecto. Tras la primera reunión con el tutor, se decidió que cada *sprint* duraría una semana y se correspondería con una serie de objetivos planteados y unos resultados a conseguir.

A continuación se describe semana por semana dichos objetivos y los resultados que se fueron consiguiendo. Se incluye adicionalmente un *sprint* inicial que transcurre entre la asignación del proyecto y el comienzo oficial.

Sprint 0: hasta el 3 marzo

Esta iteración tiene consideraciones especiales, puesto que desde que se asignaron los proyectos hasta que realmente se comenzó a trabajar semana a semana con el tutor, se realizó una búsqueda de información relacionada con la futura aplicación.

Para ello se leyeron los apuntes de la asignatura *Procesadores del Lenguaje*, se buscó información sobre programas similares y se recabó información sobre los programas que iban a ser de utilidad a la hora de desarrollar el proyecto.

Así, se creó el repositorio en *Github* y se comenzó a realizar la programación temporal utilizando el software online *VersionOne*.

Sprint 1: 3 marzo - 10 marzo

La primera reunión con el tutor fue el 3 de marzo. En ella, se habló sobre cómo se podría distribuir mejor la aplicación que se iba a crear. Se decidió entonces optar por dos soluciones: Un prototipo de línea de comandos y una aplicación similar al prototipo con una interfaz gráfica accesible.

Planteados los objetivos generales del proyecto, se empezó a trabajar en el proyecto actual.

Se planteó el primer objetivo, que fue empezar a familiarizarse con las clases de *Burgram*. Para facilitar dicha familiarización con la estructura de clases de *Burgram* y de *PLQuiz*, el tutor me recomendó instalar un plugin que de forma automática generase diagramas *UML* con la arquitectura de las aplicaciones. Este plugin sería: <http://www.objectaid.com/class-diagram>.

Otro de los objetivos planteados fue crear un prototipo de línea de comandos, concretamente la idea sería tener una pequeña aplicación en modo texto que:

1. Tome como argumento el nombre de un archivo con la especificación de una gramática en el formato *bison*.
2. Utilice su contenido para, utilizando las clases adecuadas de *Burgram*, instanciar un objeto *Gramática*.
3. Interrogue el objeto *Gramática* para mostrar el *first* y el *follow* de los símbolos de la gramática.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 2: 10 marzo - 17 marzo

El 10 de marzo se realizó una reunión en la que se valoraron los resultados del primer *sprint*. Todos los resultados fueron satisfactorios, se consiguió crear un prototipo en el cual, introduciendo una gramática en formato *bison*, se obtenía de ella el *first* y el *follow* por pantalla.

En esta reunión se plantearon nuevos objetivos:

- Generar la tabla de análisis sintáctico predictivo (TASP) en el prototipo.
- Instalar *Miktex*.
- Instalar *TexMaker*.
- Hacer un uso correcto de los Path.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 3: 17 marzo - 31 marzo

El 17 de marzo se realizó la siguiente reunión en la que se valoraron los resultados del segundo *sprint*. Todos los resultados fueron satisfactorios. Se consiguió generar la TASP y se preparó el prototipo para que los path no diesen problemas en los diferentes sistemas operativos.

Los objetivos que se plantearon en esta reunión fueron:

- Introducir parámetros en el prototipo.
- Investigar sobre diferentes motores de plantillas, como por ejemplo *Mustache*, *Trimou*, *JinJava*, *FreeMaker* o *Shootout*, y decidir cuál era el óptimo para el proyecto.
- Generar una plantilla para crear ficheros .xml utilizables por moodle.
- Generar una plantilla para crear ficheros .tex.

Se estableció que este *sprint* sería de dos semanas debido a la Semana Santa.

Terminada la reunión, el trabajo que se realizó a lo largo de las semanas y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 4: 31 marzo - 7 abril

El 31 de marzo se realizó la siguiente reunión en la que se valoraron los resultados del tercer *sprint*:

- Se comenzó el tratamiento de datos utilizando una librería llamada *Apache Commons CLI*.
- Se decidió utilizar como motor de plantillas *JinJava*.
- En cuanto a las plantillas, y dado que su generación no había sido tan satisfactoria como se pretendía, se decidió junto con el tutor, centrar el trabajo en la plantilla .xml y dejar la plantilla .tex para un *sprint* futuro.

En esta reunión se plantearon nuevos objetivos:

- Introducción de *JinJava* para comenzar con la generación de los ficheros deseados.
- Continuar con la plantilla para ficheros .xml.
- Establecer una codificación *UTF-8* para todos los ficheros, tanto en las clases del proyecto como en los ficheros generados.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 5: 7 abril - 14 abril

La siguiente reunión tuvo lugar el 7 de abril y se analizaron los progresos conseguidos en el anterior *sprint*. Se desechó el uso de *JinJava* como motor de plantillas porque no permitía la posibilidad de realizar bucles de repetición. En su lugar, se decidió continuar el trabajo utilizando *mustache*, ya que cualquier elemento iterable puede ser utilizado. En cuanto a la plantilla para ficheros .xml, no se pudo avanzar lo deseado puesto que aparecieron problemas con *JavaScript* ya que la plataforma moodle de la Universidad de Burgos no permite el su uso.

Los objetivos que se plantearon en esta reunión fueron:

- Al tener problemas con el uso de *JavaScript*, se plantearon dos posibles soluciones. Una de ellas era utilizar un motor web que, desde Java, permitiese transformar un documento html construido de forma dinámica

con *JavaScript*, en un documento html estático. La otra solución era cambiar todos los script de *JavaScript* por html estático de forma manual. Por tanto, el objetivo era buscar dicho motor web. Si se conseguía encontrar un motor web que transformase los documentos en poco tiempo el objetivo estaría cumplido; sino, se tendría que proceder al cambio de todos los script de forma manual.

- Buscar información sobre el formato de inicio y fin de las preguntas en moodle.
- Durante la reunión se decidió que las gramáticas estuvieran compuestas por letras mayúsculas y letras minúsculas. Las mayúsculas corresponderían a los no terminales y las minúsculas a los terminales. Se estableció que no se utilizasen símbolos como paréntesis, corchetes, llaves, etc. puesto que interfieren con el funcionamiento de la plantilla. De esta manera, se planteó como objetivo cambiar las gramáticas y adaptarlas a los nuevos requisitos planteados en esta reunión.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 6: 14 abril - 21 abril

En la siguiente reunión, que tuvo lugar el 14 de abril, se analizaron los resultados del quinto *sprint*. Debido a la dificultad de encontrar un motor web que permitiese transformar documentos html dinámicos en estáticos, se decidió implantar la segunda solución planteada en la reunión anterior para solucionar el problema que suponía el uso de *JavaScript*. Para ello, se utilizó la propiedad de *mustache* que permite utilizar cualquier elemento iterable. Los demás objetivos tuvieron resultados satisfactorios.

En esta reunión se plantearon nuevos objetivos:

- Crear respuestas alternativas para los símbolos de preanálisis con cierta lógica conociendo la solución correcta.
- Continuar con la plantilla .xml.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 7: 21 abril - 28 abril

El 21 de abril se realizó una reunión en la que se valoraron los resultados del sexto *sprint*. Todos los resultados fueron satisfactorios.

Ya que todos los objetivos se iban cumpliendo a tiempo, en esta semana se planteó terminar el código para visualizar las tablas del *first* y del *follow* en una gramática *LL*.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 8: 28 abril - 5 mayo

En la siguiente reunión celebrada el 28 de abril se comprobó que los objetivos del séptimo *sprint* se habían conseguido, es decir, el código para visualizar las tablas del *first* y del *follow* en una gramática *LL* estaba terminado.

En esta reunión se plantearon nuevos objetivos:

- Se estableció cómo debería ser la plantilla y se programó cambiar la plantilla existente en ese momento para que fuera acorde con lo decidido en la reunión.
- Crear la tabla de análisis sintáctico predictivo introduciendo la respuesta correcta. Crear un método para obtener las cadenas de múltiples respuestas. Todo ello para la gramática *LL*.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 9: 5 mayo - 12 mayo

En la reunión del 5 de mayo se valoraron los resultados obtenidos en el octavo *sprint*. Se comprobó que para la gramática *LL* todo estaba correcto y terminado, menos la tabla de la traza, que se decidió dejar para un futuro *sprint*. Aún así, se decidió meter un parámetro en el prototipo con la cadena para usar posteriormente.

Siguiendo los pasos de la gramática *LL* se programó para este *sprint* comenzar a generar las gramáticas *LR*, *SLR* y *LALR*. Además, se comentó que se debería comenzar a redactar y escribir las partes de la memoria que ya estuvieran definidas hasta ese momento.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 10: 12 mayo - 19 mayo

En la siguiente reunión, del 12 de mayo, se comprobó que se habían generado las tablas de conjuntos de la gramática *LR* pero que todavía faltaban por generar las tablas correspondientes a las otras dos gramáticas.

En esta reunión se planteó:

- Crear un método para generar respuestas cortas en moodle.
- Crear un método para obtener la *tabla de acción e Ir a*.
- Seguir trabajando en las gramáticas restantes.
- Seguir trabajando en la memoria.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 11: 19 mayo - 26 mayo

El 19 de mayo, en la siguiente reunión se comprobó que se habían generado correctamente las respuestas cortas, así como la *tabla de acción e Ir a* en todas las gramáticas. Además se presentó un primer avance de la memoria. Se comprobó además que el cambio en el posicionamiento de los conjuntos *LR*, *SLR* y *LALR* era correcto.

El avance de la aplicación por línea de comando era correcto y quedaban pocas cosas por ir terminando, por tanto, se planteó empezar a desarrollar la interfaz gráfica. Además, y dado que las gramáticas estaban prácticamente finalizadas, se programó terminarlas creando la tabla para la traza en todas ellas. Por último, el tutor me facilitó plantillas de memorias de años anteriores para tener modelos a seguir y comenzar con la redacción de los anexos.

Terminada la reunión, el trabajo que se realizó a lo largo de la semana y las horas invertidas en él fueron las siguientes:

FALTA

Sprint 12: 26 mayo - 2 junio

En la reunión del 26 de mayo se valoraron los resultados del undécimo *sprint* y se comprobó que todos eran satisfactorios.

Aún avanzando a buen ritmo y con la mayoría del proyecto terminado, se decidió en esta reunión presentarlo en la segunda convocatoria para tener tiempo y terminar de pulir aquellas cosas que todavía no estaban del todo correctas.

En esta reunión se comprobó que, en líneas generales, únicamente quedaba por hacer para la finalización del proyecto y, por tanto, serían los objetivos del doceavo *sprint*:

- Crear pruebas unitarias para evaluar el funcionamiento de los métodos del prototipo.
- Terminar la interfaz gráfica.
- Retomar y terminar la plantilla .tex.
- Seguir trabajando en la memoria y en sus anexos.

Sprint 13: 2 junio - 9 junio

En la siguiente reunión, que tuvo lugar el 2 de junio se comprobaron los avances realizados en el anterior *sprint*. Se comprobó que se habían creado correctamente las pruebas unitarias para evaluar el funcionamiento de los métodos del prototipo. Dichas pruebas se han completado con una cobertura suficiente dadas las características del proyecto. Además, se presentaron los avances conseguidos en la plantilla .tex para los análisis *LL* y se plantearon diferentes alternativas para mejorar la forma de la misma.

Como objetivos para el siguiente *sprint*, se plantearon:

- Continuar con la plantilla .tex para el resto de análisis e introducir las mejoras planteadas en la reunión.
- Terminar la interfaz gráfica.
- Empezar a pensar en la forma y contenido del póster necesario para la defensa del proyecto.
- Seguir trabajando en la memoria y en sus anexos.

Sprint 14: 9 junio - 16 junio

Sprint 15: 16 junio - 23 junio

Sprint 15: 23 junio - 30 junio

A.3. Estudio de viabilidad

El análisis de viabilidad es un requisito imprescindible en cualquier organización de una proyecto, sea éste del tipo que sea.

Son muchos los aspectos que intervienen a la hora de valorar la viabilidad de un proyecto y no es recomendable centrarse únicamente en la relación coste-beneficio del mismo, sino que conviene valorar otros aspectos como la funcionalidad, el tamaño del proyecto, el grado de complejidad del mismo, el equipo de trabajo, etc.

En este caso, y dado la finalidad del proyecto actual, un estudio de viabilidad complejo no es necesario, puesto que el proyecto está planteado dentro de un contexto docente y al amparo de la Universidad de Burgos.

De todos modos, se plantean algunos aspectos generales, dividiendo el estudio de viabilidad en los dos criterios que se han considerado más importantes: la viabilidad económica y viabilidad legal.

Viabilidad económica

Debido a la estructura del proyecto actual, no es necesario realizar un estudio de costes del mismo, ya que el proyecto está realizado como una actividad docente más dentro del Grado en Ingeniería Informática y sus futuras aplicaciones quedarán también ubicadas en este ámbito.

Además, se trata de una aplicación uso gratuito dentro de la Universidad de Burgos.

Viabilidad legal

En el uso de bibliotecas de terceros, como por ejemplo *Apache Commons Cli*, todo el software producido por la ASF (*Apache Software Foundation*) o cualquiera de sus proyectos, está desarrollado bajo los términos de la licencia *Apache*, es decir, la licencia permite al usuario del software la libertad de usar dicho software para cualquier propósito, para distribuirlo, modificarlo y distribuir versiones modificadas del software, bajo los términos de la licencia; sin necesidad de preocuparse por problemas legales futuros.[?]

Especificación de Requisitos

B.1. Introducción

Este anexo tiene como objetivo analizar y documentar las necesidades funcionales mínimas que deben ser soportadas por la aplicación desarrollada.

Otro objetivo básico del presente anexo es la de priorizar cada una de las funcionalidades a implementar. Si se tiene claro cuáles son las tareas críticas, se podrá llevar a cabo un enfoque más preciso en ellas y asignarlas una mayor cantidad de recursos.

El anexo se estructura presentado una lista de objetivos generales que la aplicación debe cumplir. Son los objetivos que el cliente establece mediante un lenguaje neutral y es trabajo del ingeniero su desglose y formalización. A continuación se incluye un catálogo de los requisitos que deberá cumplir la aplicación.

B.2. Objetivos generales

El desarrollo de la aplicación cumple los siguientes objetivos:

1. Desarrollo de una interfaz gráfica.
2. Procesado y manipulación de gramáticas mediante procesadores del lenguaje.
3. Obtención del *first* y el *follow* de una gramática.
4. Obtención de la tabla de la tabla de análisis sintáctico predictivo (*TASP*) de una gramática.
5. Obtención de los conjuntos de items $LR(0)$ y $LR(1)$.

6. Obtención de las tablas de análisis sintáctico *tabla de acción e Ir a* para análisis *SLR(1)*, *LR(1)* y *LALR(1)*
7. Exportación de cuestionarios en formato *XML* compatible con la plataforma Moodle.
8. Exportación de cuestionarios en formato *L^AT_EX*.
9. Creación de un manual de usuario detallando todas las opciones de la aplicación

B.3. Catálogo de requisitos

El objetivo de este apartado es definir de forma clara, precisa, completa y verificable todas las funcionalidades y restricciones del sistema a construir.

Funciones del producto

Una de las características más importantes de la aplicación que se ha desarrollado es su facilidad de uso. Dado que su empleo está orientado a convertirse en una herramienta docente, debe ser fácil de usar y posibilitar la creación de cuestionarios de manera rápida, sencilla y comprensible.

Como se explica en el *Anexo E: Manual de usuario*, los archivos obtenidos de la aplicación tendrán extensiones *.TEX* y *.XML*, tratando de cubrir un mayor abanico de posibilidades de uso de la aplicación y dando una mayor cobertura a la hora de obtener los cuestionarios.

En el caso de los archivos *.XML* es necesario el uso de la plataforma moodle, que al ser internacional, se puede utilizar en varios idiomas, dándole así una mayor difusión a la aplicación.

Requisitos de usuario

El usuario debe ser capaz de utilizar la aplicación mediante la interfaz gráfica o mediante la consola del sistema.

La ayuda será un punto a tener en cuenta en el caso de utilizar la aplicación mediante la consola del sistema.

En todo momento el usuario podrá consultar el Manual del Usuario para orientarse y resolver dudas que le puedan surgir durante el manejo de la misma.

Requisitos del sistema

Los requisitos mínimos del hardware del sistema serán:

- Intel core i3.

- 128 MB de memoria RAM.
- Resolución de pantalla igual o superior a 1152 x 864, solo para la GUI.

Se recomienda trabajar con unos requisitos concretos de hardware para que el funcionamiento de la aplicación sea el mejor posible. Estos requisitos recomendados serán:

- Intel core i7.
- 1 GB de memoria RAM.
- Resolución de pantalla igual o superior a 1280 x 960 , solo para la GUI.

Los requisitos software necesarios son:

- Sistema operativo *Windows*.
- Máquina virtual de java (*JDK8*).
- Consola de sistema operativo capaz de interpretar Unicode, solo para la interfaz en línea de comandos. Si no soportara Unicode, las tildes u otros símbolos podrían ser mal representados.

B.4. Especificación de requisitos

En este apartado se realizará un análisis de los requisitos de diseño de la aplicación. Para ello, se detallarán los requisitos funcionales y no funcionales de la aplicación.

Requisitos funcionales

La aplicación debe ser capaz de cumplir los siguientes requisitos:

- *RF1: Cargar gramática y análisis.* Permite al usuario seleccionar una gramática disponible en la carpeta *GRAMÁTICAS* y un tipo de análisis.
- *RF2: Obtener análisis.* Se obtienen los datos del análisis de la gramática seleccionada.
- *RF3: Crear cadenas.* Se obtienen las cadenas de los datos del análisis.
- *RF4: Obtener fichero .TEX.* Se utiliza la plantilla para obtener el fichero *.TEX*.
- *RF5: Obtener fichero .XML.* Se utiliza la plantilla para obtener el fichero *.XML*.

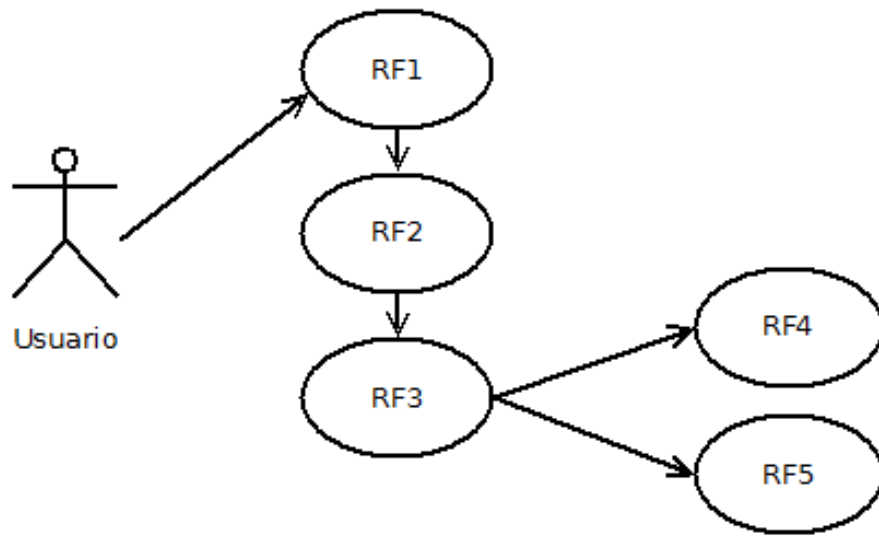


Figura B.1: Diagrama principal de casos de uso

Requisitos no funcionales

A continuación se describen todos aquellos requisitos no funcionales importantes para considerar en el diseño:

- *RNF1: Eficiencia.* Se debe buscar la eficiencia de la aplicación para que pueda ser usada con gramáticas complejas.
- *RNF2: Extensibilidad.* Se pueden definir nuevas capacidades en la aplicación mediante su extensión a otro tipo de ficheros.

Especificación de diseño

C.1. Introducción

En este anexo se detallan todos los aspectos del diseño basándose en el análisis realizado en el Apéndice B, buscando una solución a medida para los problemas detectados.

La frontera entre la finalización del análisis y el comienzo del diseño es difusa ya que el modelo evoluciona y se refina en cada paso.

El diseño es fundamental para el correcto desarrollo de un proyecto software porque facilita la estructuración modular, identificando cada elemento del programa y facilitando su uso.

En el diseño del proyecto se pueden diferenciar varias fases:

1. Diseño del tipo abstracto de datos para instanciar los objetos.
2. Diseño de los algoritmos para la obtención del resultado deseado.
3. Diseño de las plantillas para generar los ficheros .tex y .xml.
4. Diseño de la interfaz de usuario, que se desarrolla de dos formas diferentes:
 - Interfaz en línea de comandos.
 - Interfaz gráfica de usuario.

C.2. Diseño de datos

El objetivo de este capítulo es seleccionar las representaciones lógicas de los datos (estructuras de datos) que se han identificado en la fase de análisis.

Para la representación se utilizarán diagramas de clases que permiten modelar el diseño de la aplicación.

Se ha intentado conseguir la mayor claridad posible a la hora de representar las clases a generar. Debido a la gran cantidad de clases generadas, se ha optado por no representarlas todas, intentando agrupar por funcionalidad y representando tan sólo aquellas que aporten información y valor al diagrama en el que se representan.

Para facilitar la legibilidad de las mismas, se ha optado por no mostrar algunos de los atributos y métodos de algunas de las clases.

Los diagramas de clases del proyecto actual están basados en los obtenidos del programa *BURGRAM*. Los diagramas de *PLGRAM* son una modificación de los anteriores de manera que el trabajo ha desarrollar fuera más sencillo y eficaz. A continuación, se muestran algunos de ellos y una pequeña explicación de los mismos.

Estructura de las gramáticas

El caso de la Estructura de las gramáticas, se muestra en la Figura C.1. En ella, se muestran las relaciones y herencias entre las clases.

Puede observarse que la *Gramática* está compuesta por un *Vector de Producciones*, que a su vez está compuesto por una o varias *Producciones*. Dichas *Producciones* está compuestas por un *Vector de Símbolos*, formado por *Símbolos* o un *No Terminal*. Los *Símbolos* pueden ser *Terminales*, *No Terminales* o *Nulos*.

A su vez, la gramática tiene un *First* y un *Follow*, que pueden estar compuestos por un *Vector Símbolos*.

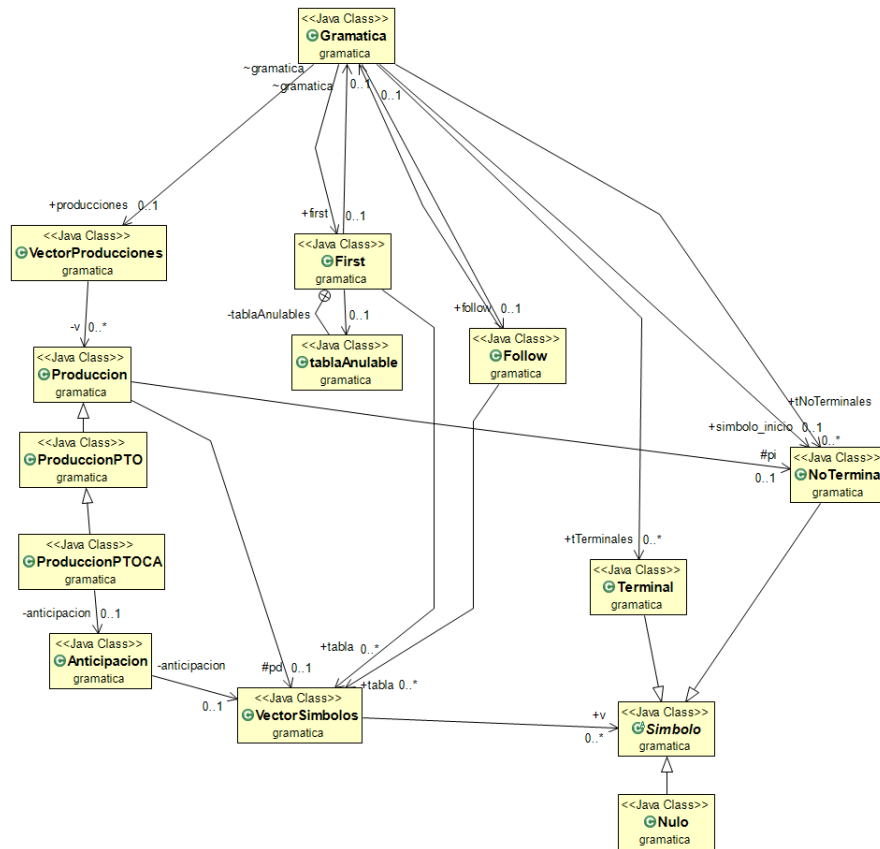


Figura C.1: Diagrama de clases: Estructura de las gramáticas

Estructura de las tablas de análisis

En la Figura C.2 se muestra la Estructura de las tablas de análisis. Puede observarse que las clases *TablaDescendente* y *TablaAscendente* son subclases de la clase *Tabla* y, por lo tanto, heredan sus métodos.

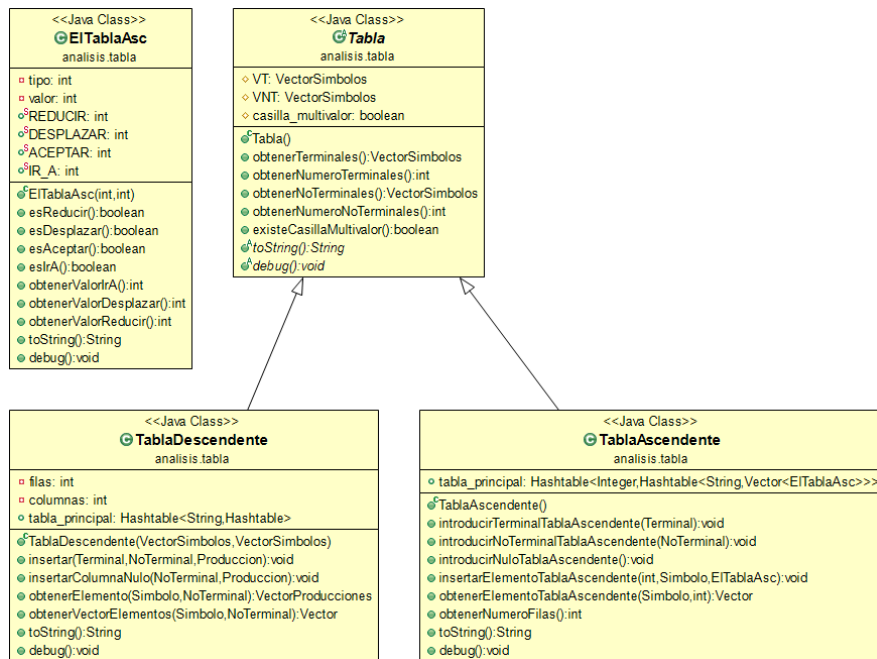


Figura C.2: Diagrama de clases: Estructura de las tablas de análisis

Estructura del informe del análisis

En este Diagrama de clases, se observa que las clases *InformeAscendente* e *InformeDescendente* heredan de la clase *Informe*, y por tanto, heredan sus métodos.

Existe también una clase interna de la clase *Informe*, denominada *Plantilla*.

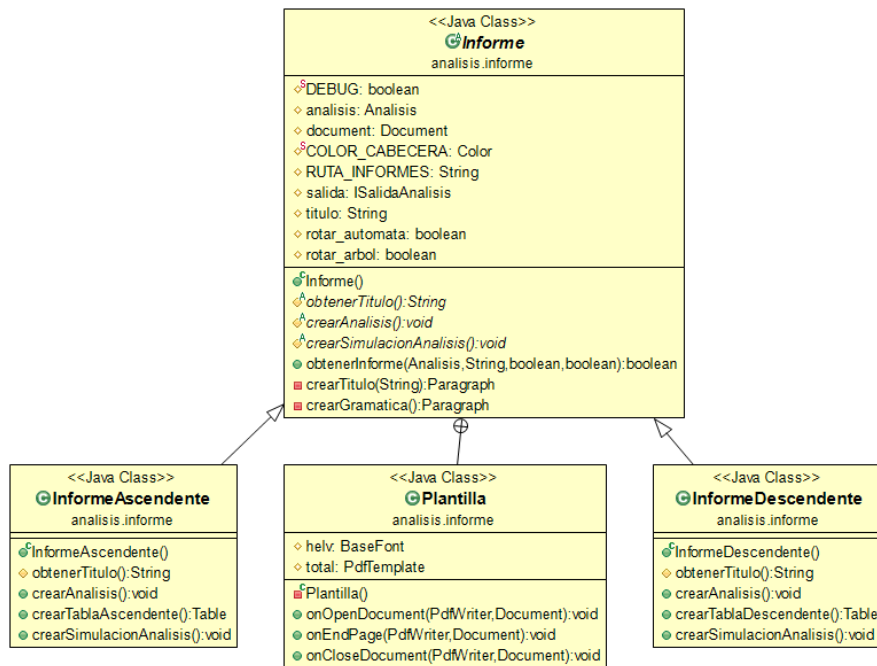


Figura C.3: Diagrama de clases: Estructura del informe del análisis

Estructura de las clases del análisis

En el Diagrama de clases mostrado en la Figura C.4 se tiene una clase abstracta *Analisis*, de la cuál heredan las clases *AnalisisLL1* y *AnalisisSLR1*. La clase *AnalisisSLR1* tiene como subclases las clases *AnalisisLR1* y *AnalisisLALR1*.

Además, se puede observar en el Diagrama la interfaz *ISalidaAnalisis* que aporta un conjunto de métodos comunes para todas las clases posteriores, en este caso, las comentadas anteriormente.

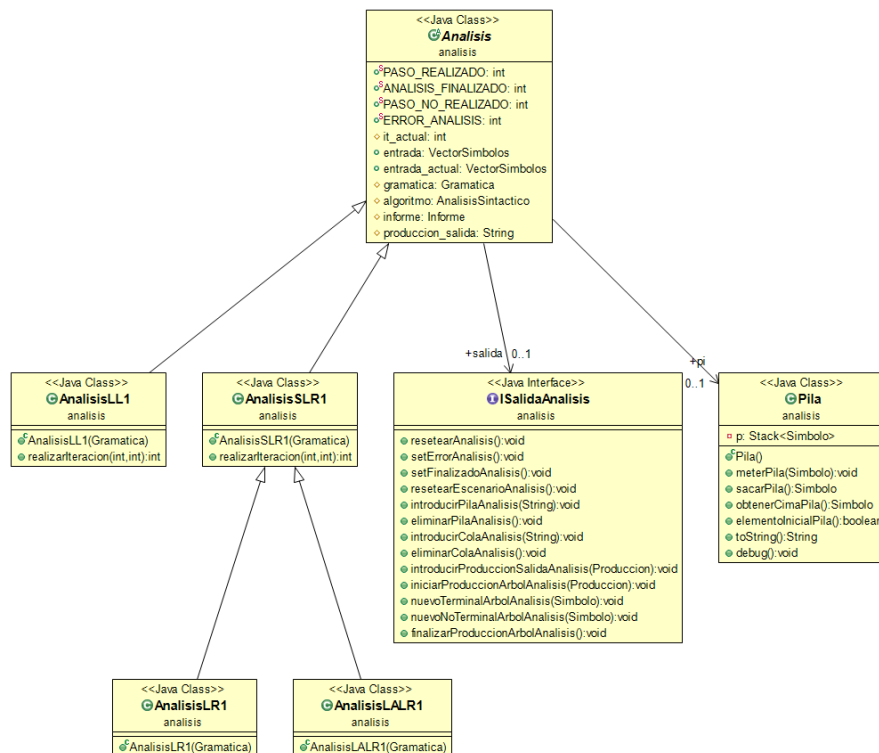


Figura C.4: Diagrama de clases: Estructura de las clases del análisis

Estructura del prototipo

La clase *Prototipo* está compuesta por diferentes métodos que permiten crear distintas cadenas dependiendo si se quiere un fichero .xml o .tex.

La clase *Prototipo* tiene diferentes clases internas que permiten guardar la estructura de datos usada en la plantilla. Estas clases internas son las que se muestran en la parte baja del Diagrama de la Figura C.5.

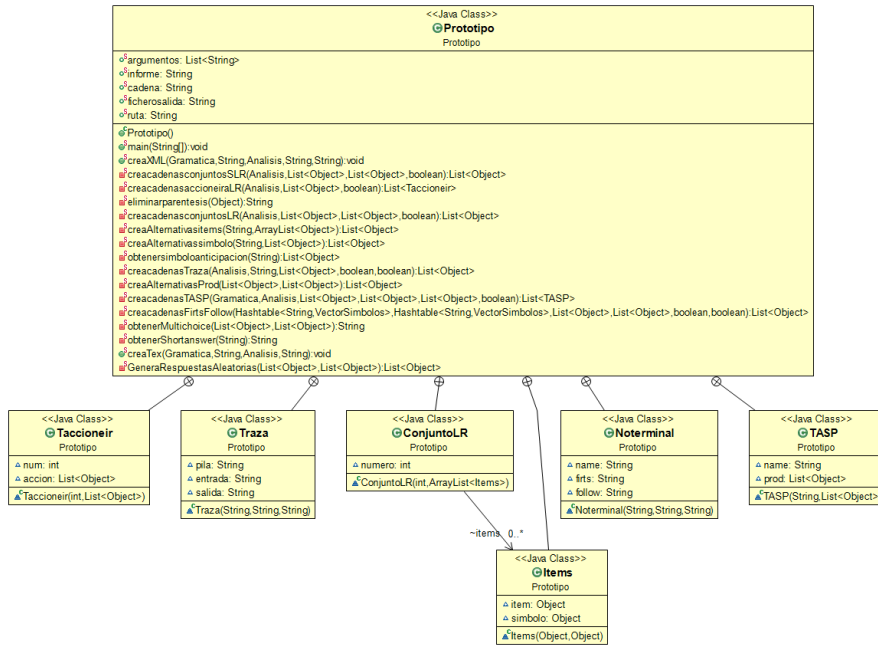


Figura C.5: Diagrama de clases: Estructura del prototipo

C.3. Diseño procedimental

En esta sección deberían aparecer los casos de uso más complejos que requieren un análisis más exhaustivo y pormenorizado.

Sin embargo, el proyecto desarrollado, no ha requerido casos de uso complejos, sino que ha sido un tratamiento de cadenas y un almacenamiento de datos en clases con una estructura especial. Estas clases son las clases internas que se han comentado en el Diagrama de clases: Estructura del prototipo.

Todos los datos de la plantilla están guardados en un diccionario (*HashMap*) compuesto por claves y valores. Las claves son los nombres que va a utilizar la plantilla. Los valores pueden ser cadenas de caracteres o listas de estructuras de datos.

El diseño procedimental general de la aplicación es bastante sencillo. Básicamente se necesitan dos elementos, una plantilla y unos datos, que como se ha comentado anteriormente se encuentran en un *HashMap*. Bastaría con compilar la plantilla con dichos datos para obtener el fichero final deseado. Este simple procedimiento se ejemplifica a continuación:

1. Plantilla

```
{{#FirstFollow}}
```



```
Nombre: {{name}} first: {{first}} follow: {{follow}}.
{{/FirstFollow}}
```

2. Estructura de datos

Name	Value
▼ FirstFollowSin	ArrayList<E> (id=59)
▼ ▲ elementData	Object[10] (id=71)
▼ ▲ [0]	Prototipo\$Noterminal (id=72)
> ▲ first	"begin" (id=83)
> ▲ follow	"end" (id=84)
> ▲ name	"A" (id=65)
▼ ▲ [1]	Prototipo\$Noterminal (id=73)
> ▲ first	"tipo,id" (id=81)
> ▲ follow	"\$" (id=82)
> ▲ name	"S" (id=66)
▼ ▲ [2]	Prototipo\$Noterminal (id=74)
> ▲ first	"codigo" (id=79)
> ▲ follow	"end" (id=80)
> ▲ name	"C" (id=67)
▼ ▲ [3]	Prototipo\$Noterminal (id=75)
> ▲ first	"tipo,id" (id=77)
> ▲ follow	"begin" (id=78)
> ▲ name	"B" (id=68)

Figura C.6: Ejemplo de una estructura de datos

3. Resultado obtenido de la compilación de la plantilla anterior con la estructura de datos mostrada.

```
Nombre: A first: begin follow: end.
Nombre: S first: tipo,id follow: $.
Nombre: C first: codigo follow: end.
Nombre: B first: tipo,id follow: begin.
```

Evidentemente, el proyecto es más complejo de lo mostrado en el ejemplo anterior. La dificultad del mismo radica en dos pilares fundamentales:

- Creación de las estructuras de datos. El principal problema fue que las estructuras de datos eran variables según la gramática que se introducía y por tanto debían ser dinámicas. Este problema se solucionó creando las clases internas ya comentadas.

- Diseño de la plantilla. En un principio se planteó crear una plantilla genérica que sirviera para la obtención de los dos tipos de fichero. Aunque actualmente, después del desarrollo de todo el proyecto, es algo que sí podría llevar a cabo, en el momento de empezar con el diseño de las plantillas era algo que se antojaba bastante complejo. Por tanto, se optó por realizar dos diseños de plantillas, uno para cada tipo de fichero. Además, cada tipo de plantilla tenía que servir para cuatro tipos diferentes de análisis, que a su vez están formados por distintos elementos. Este complejo entramado es lo que complicó el diseño de la plantilla. Se tuvo que crear una plantilla que recopilara todos los siguientes datos:

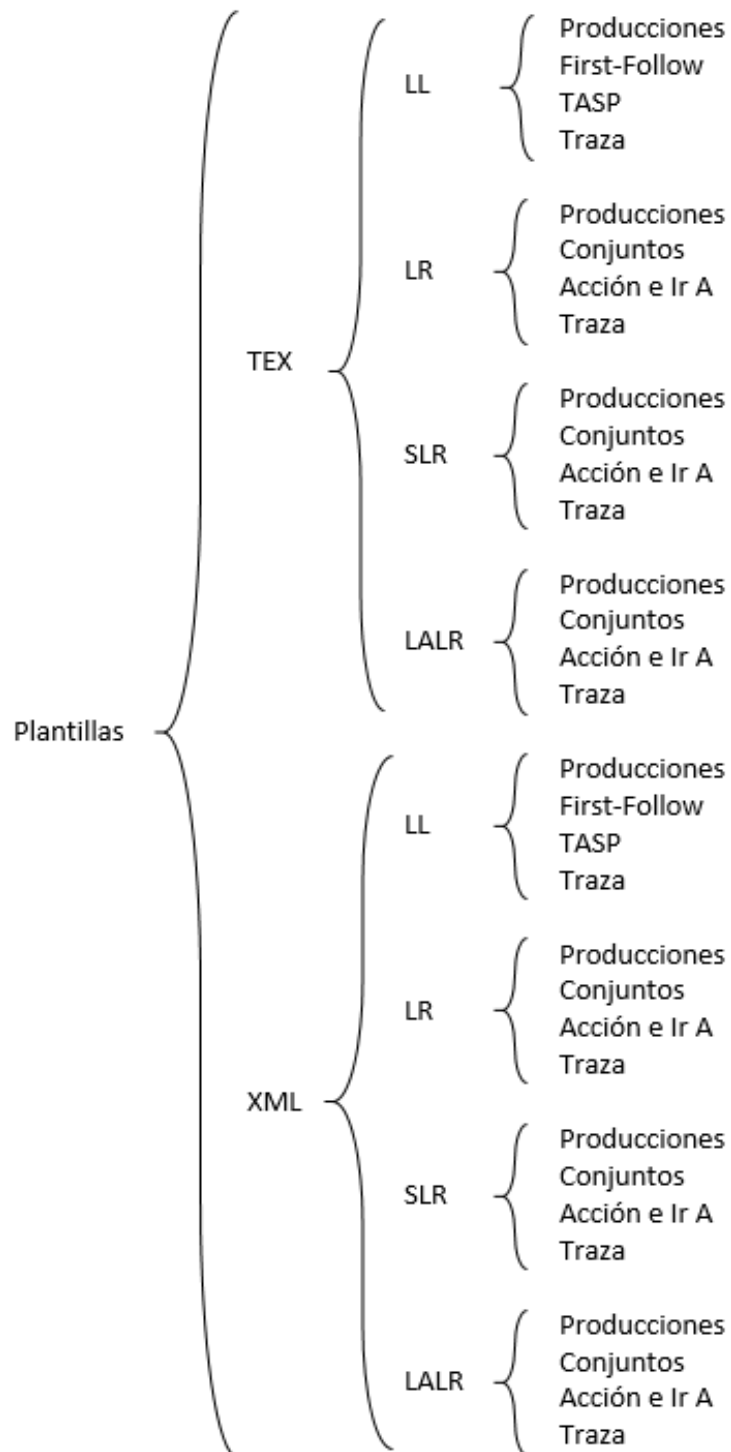


Figura C.7: Esquema general de las plantillas

C.4. Diseño arquitectónico

Documentación técnica de programación

- D.1. Introducción
- D.2. Estructura de directorios
- D.3. Manual del programador
- D.4. Compilación, instalación y ejecución del proyecto
- D.5. Pruebas del sistema

Documentación de usuario

E.1. Introducción

En esta sección se explica cómo llevar a cabo la instalación de las herramientas implementadas en el presente proyecto y los requisitos del sistema.

E.2. Requisitos de usuarios

Los requisitos mínimos necesarios son los siguientes:

Requisitos hardware

Requisitos mínimos:

- Intel core i3.
- 128 MB de memoria RAM.
- Resolución de pantalla igual o superior a 1152 x 864, solo para la GUI.

Requisitos recomendados:

- Intel core i7.
- 1 GB de memoria RAM.
- Resolución de pantalla igual o superior a 1280 x 960 , solo para la GUI.

Requisitos software

Los requisitos software necesarios son:

- Sistema operativo *Windows* o *Linux*.
- Máquina virtual de java (*JDK8*).
- Consola de sistema operativo capaz de interpretar Unicode, solo para la interfaz en línea de comandos. Si no soportara Unicode, las tildes u otros símbolos podrían ser mal representados.

E.3. Instalación

La aplicación se distribuye de dos formas diferentes: *PLGRAM* y *PLGRAMlineCommand*:

- Para ejecutar *PLGRAM* se dispone de un fichero jar ejecutable.

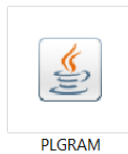


Figura E.1: Icono PLGRAM

- Para ejecutar *PLGRAMlineCommand* se debe acceder mediante la línea de comandos a la carpeta que contiene el fichero.

Esta carpeta debe contener también una carpeta llamada *gramáticas* donde estarán las gramáticas guardadas. Además, también estarán en ese directorio las plantillas necesarias.

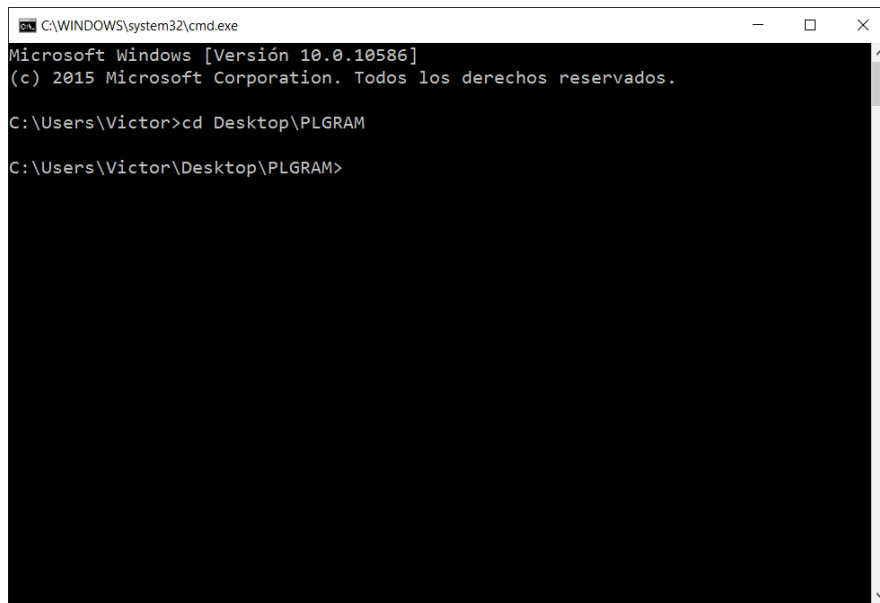


Figura E.2: Acceso a la carpeta contenedora de *PLGRAM* desde la consola

E.4. Manual del usuario

Aplicación línea de comandos

Para ejecutar la aplicación se debe escribir el siguiente comando:

```
java -jar PLGRAMlineCommand.jar -g -t [-i] [-ca] [-o]
```

Los diferentes argumentos son:

- *-g* Nombre de la gramática a analizar.
- *-t* Tipo de análisis a realizar [*LL*, *LR*, *SLR*, *LALR*].
- *-i* Extensión del informe [*TEX*, *XML*, *ALL*].
- *-ca* Cadena a analizar.
- *-o* Nombre del fichero de salida.

La gramática *-g* y el tipo de análisis a realizar *-t* son obligatorios. La gramática debe estar en la carpeta gramáticas.

El parámetro *-t* tiene diferentes opciones:

- *LL* Para realizar un análisis *LL*.

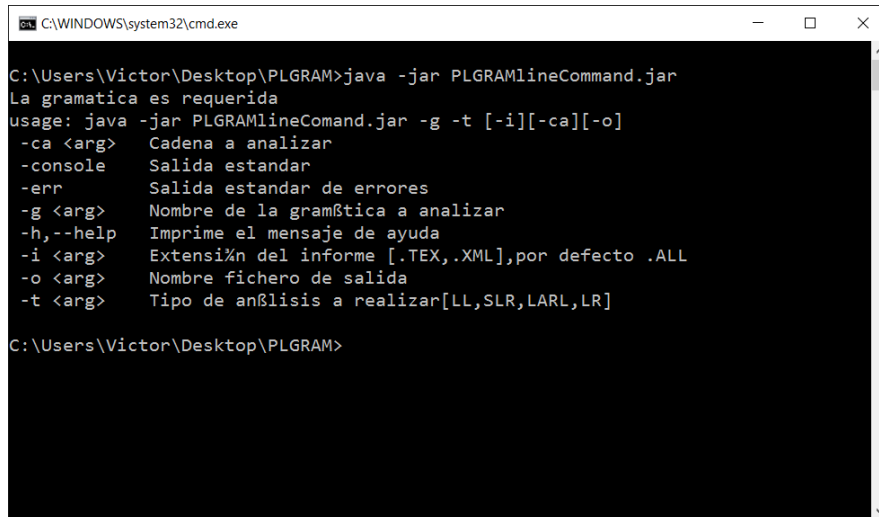
- *LR* Para realizar un análisis *LR*.
- *SLR* Para realizar un análisis *SLR*.
- *LALR* Para realizar un análisis *LALR*.

Si no se introduce ningún parámetro se indicará que falta el parámetro de la gramática, ya que como se ha comentado anteriormente, es un parámetro obligatorio.

Así, si se introduce la siguiente línea de código

```
java -jar PLGRAMlineCommand.jar
```

La aplicación requerirá la gramática, tal y como se muestra a continuación:



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop\PLGRAM>java -jar PLGRAMlineCommand.jar
La gramatica es requerida
usage: java -jar PLGRAMlineComand.jar -g -t [-i][-ca][-o]
       -ca <arg>  Cadena a analizar
       -console    Salida estandar
       -err        Salida estandar de errores
       -g <arg>   Nombre de la gramática a analizar
       -h,--help   Imprime el mensaje de ayuda
       -i <arg>   Extensi n del informe [.TEX,.XML],por defecto .ALL
       -o <arg>   Nombre fichero de salida
       -t <arg>   Tipo de an lisis a realizar[LL,SLR,LARL,LR]

C:\Users\Victor\Desktop\PLGRAM>
```

Figura E.3: *PLGRAM* sin argumento gramática

La misma situación se presenta con el parámetro tipo de análisis a realizar.

Si se introduce la línea de código

```
java -jar PLGRAMlineCommand.jar -g gramatica1.yc
```

en la que no se ha especificado el tipo de análisis, la aplicación lo solicita, como se muestra a continuación:

```

C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop\PLGRAM>java -jar PLGRAMlineCommand.jar -g gramatica1.yc
El tipo de analisis a realizar es requerido
usage: java -jar PLGRAMlineComand.jar -g -t [-i][-ca][-o]
       -ca <arg> Cadena a analizar
       -console  Salida estandar
       -err      Salida estandar de errores
       -g <arg>  Nombre de la gramática a analizar
       -h,--help Imprime el mensaje de ayuda
       -i <arg>  Extensi n del informe [.TEX,.XML],por defecto .ALL
       -o <arg>  Nombre fichero de salida
       -t <arg>  Tipo de an lisis a realizar[LL,SLR,LARL,LR]

C:\Users\Victor\Desktop\PLGRAM>

```

Figura E.4: *PLGRAM* sin argumento tipo de análisis

Si se introducen los parámetros de la gramática y del tipo de análisis que se quiere realizar, se ejecuta el programa con el resto de valores por defecto:

- *-i* con valor *ALL*.
- *-o* que se denominará de manera genérica como el tipo de análisis elegido concatenado al nombre de la gramática seleccionada. Por ejemplo, para un análisis *LL* y una gramática *G3*, los ficheros generados se denominarán *LLG3*, cada uno con su extensión correspondiente.
- *-ca* no se generará ninguna tabla de primeros pasos de la traza de análisis.

En el supuesto de que se quiera modificar alguno de los valores que por defecto se les asignan a los parámetros anteriores, bastará con introducir en cada uno de ellos los argumentos deseados:

- *-i* se podrá elegir la extensión del fichero de salida, que será *XML* en el caso de querer obtener un cuestionario virtual o *TEX* en el caso de que se quiera para una futura impresión.
- *-o* se podrá nombrar los ficheros de salida según se desee.
- *-ca* se podrá introducir una cadena para que la aplicación genere una tabla de primeros pasos de la traza de análisis de dicha cadena. La cadena deberá introducirse entrecomillado y cada ítem separado por un espacio. Por ejemplo, *-ca "item1 item2 item3 item4"*.

Una vez introducidos todos los parámetros, se muestra la información de la gramática elegida a través de la consola.

A continuación se muestran los datos obtenidos al introducir una gramática *gramatica1.yc* y un tipo de análisis *LL*. El resto de parámetros se han dejado por defecto.

```

C:\WINDOWS\system32\cmd.exe
C:\Users\Victor\Desktop\PLGRAM>java -jar PLGRAMlineCommand.jar -g gramatica1.yc -t LL
gramatica: gramatica1.yc

TERMINALES
    end,tipo,begin,codigo,id
NO TERMINALES
    A,S,C,B
SIMBOLO INICIO
    S
PRODUCCIONES
S->B A end
A->begin C
C->codigo
B->tipo
B->id B

first
C{codigo}
B{tipo id}
id{id}
A{begin}
begin{begin}
end{end}
S{tipo id}
tipo{tipo}
codigo{codigo}

follow
A{end}
S{S}
C{end}
B{begin}

Fichero.TEX generado correctamente en C:\Users\Victor\Desktop\PLGRAM\LLgramatica1.yc.tex
Fichero.XML generado correctamente en C:\Users\Victor\Desktop\PLGRAM\LLgramatica1.yc.xml
C:\Users\Victor\Desktop\PLGRAM>

```

Figura E.5: *PLGRAM* Con argumentos gramática y tipo de análisis

Como se puede observar, al final de la información obtenida, la consola nos muestra que los ficheros se han generado correctamente y su ruta.

Aplicación interfaz gráfica

FALTA

Ficheros obtenidos

Fichero .XML

El fichero *XML* obtenido tiene una estructura general común sea cual sea la gramática y el tipo de análisis realizado. Esta estructura sigue el siguiente esquema general:

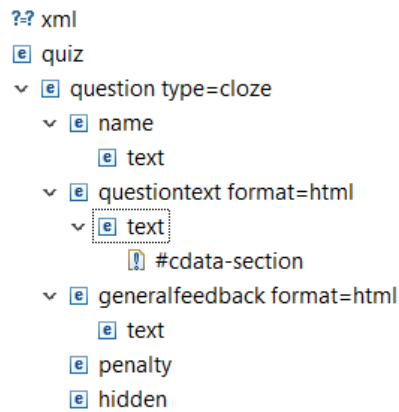


Figura E.6: Estructura del fichero.XML

Para poder utilizar el fichero.XML obtenido es necesario poder acceder a la plataforma de aprendizaje de moodle.

Los pasos que se deben seguir son:

1. Ir a la página principal del curso.
2. Hacer clic en Banco de preguntas desde el bloque de Administración.
3. Pulsar en Importar archivos.
4. Seleccionar la opción de formato XML Moodle.
5. Desde la siguiente ventana pulsar en Examinar.
6. Seleccionar el fichero.XML en nuestro ordenador.
7. Pulsar en Importar este archivo.

Los puntos 5 y 6 se pueden realizar arrastrando el archivo al recuadro que se muestra.

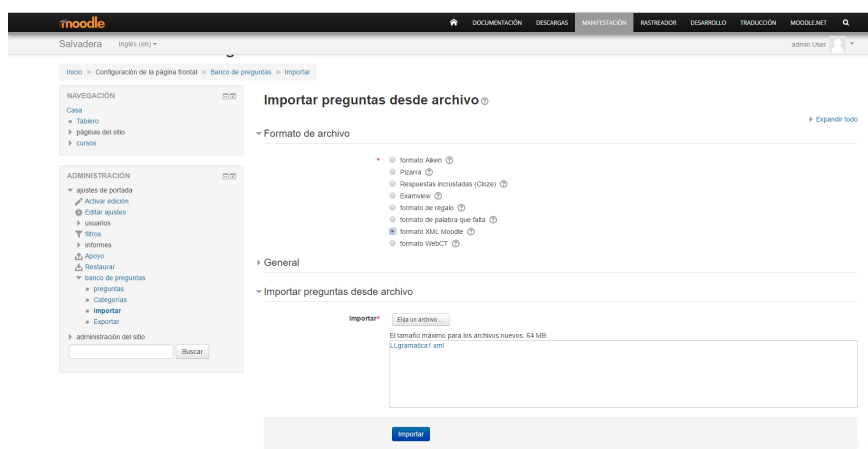


Figura E.7: Importar fichero.XML en moodle

A la hora de importar el fichero.XML hay que tener en cuenta lo siguiente:

- Los nombres de los archivos y carpetas no deben tener caracteres especiales como palabras acentuadas, tabulaciones, ñ, retornos de carro, espacios en blanco, ni símbolos de sistema como / o : , además no es recomendable combinar mayúsculas y minúsculas en los nombres.
- El tamaño del archivo no debe superar el límite permitido.

Una vez importado el fichero.XML deseado, aparece una primera previzualización de los datos que se obtendrán a continuación. Basta con dar a continuar para acceder a la siguiente pantalla.

En ella, aparecen todos los ficheros que se hayan importado. Para visualizar uno de ellos, se hace click en la lupa del archivo correspondiente.

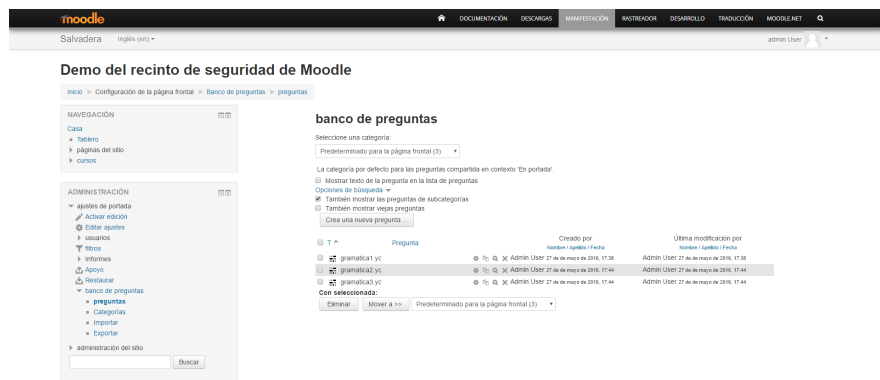


Figura E.8: Lista de ficheros.XML en moodle

Un ejemplo de cuestionario obtenido sería el siguiente:

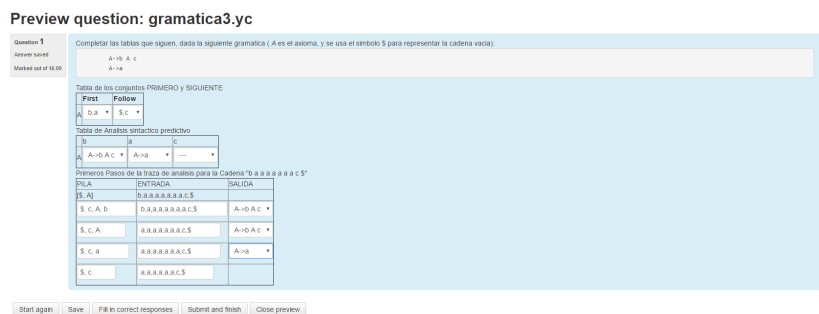


Figura E.9: Ejemplo de cuestionario resuelto en moodle

Fichero .TEX

Una vez realizado el cuestionario siguiendo cualquiera de las vías explicadas anteriormente, se obtiene un fichero .TEX en una carpeta llamada *informes* situada en el directorio donde se ejecuta el ejecutable .jar.

Al tratarse de un fichero .TEX se requiere un editor de textos adaptado a \LaTeX para utilizarlo.

Una vez abierto el fichero, su tratamiento es muy sencillo. Bastará con compilar el código y se obtendrá directamente un visionado del cuestionario que se ha realizado.

Se muestra un ejemplo de un fichero .TEX abierto con \TeX MaKer. De manera general, se puede observar el código a la izquierda y del documento generado a la derecha.

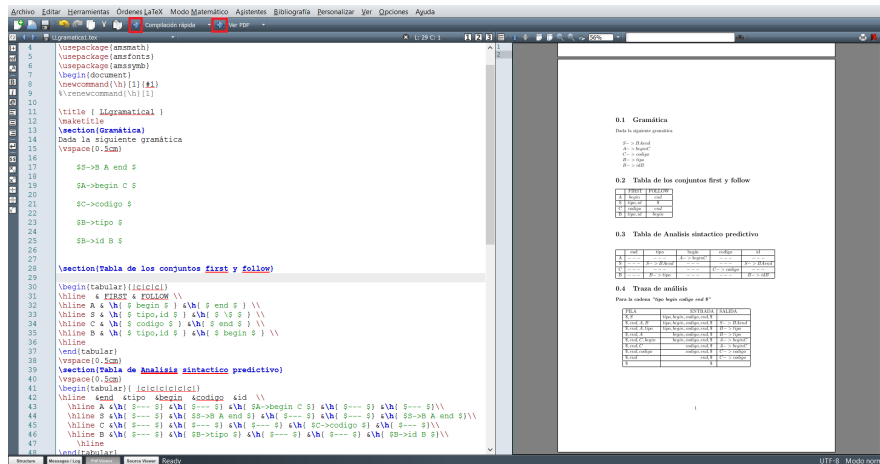


Figura E.10: Ejemplo fichero .TEX

Para una mejor comprensión, se muestra un detalle de cómo compilar el código para obtener el documento. Para ello, basta con hacer click en las flechas resaltadas en la parte superior de la pantalla del editor.

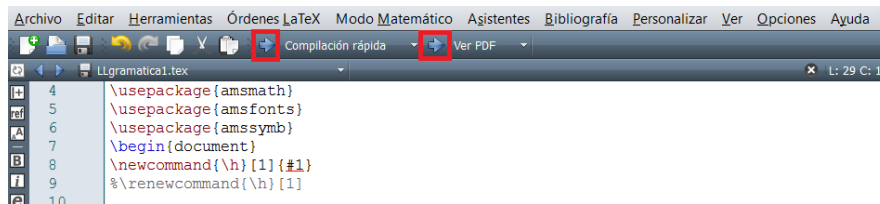


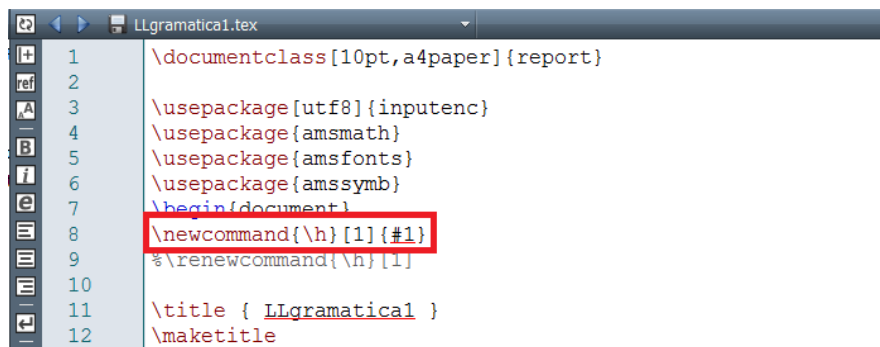
Figura E.11: Compilación de un fichero .TEX

Es necesario destacar que para cada tipo de gramática elegida, se genera un fichero .TEX diferente siguiendo los requisitos de cada una de ellas.

La aplicación ofrece dos opciones en lo que a los ficheros .TEX respecta. Así, se podrá obtener un documento en el que el cuestionario esté completo con las respuestas correctas y otro documento en el que el cuestionario se encuentre vacío, pensado para ser respondido por un tercero.

- Obtención del cuestionario completo. Es la opción que se genera por defecto y por tanto no será necesario tratar el código para su obtención. De cualquier manera, debería estar activa la siguiente línea:

```
\newcommand{\h}[1]{\hline}
```

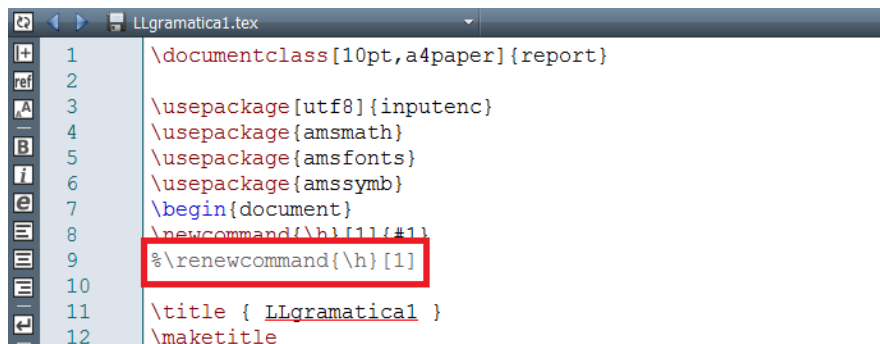


```
1 \documentclass[10pt,a4paper]{report}
2
3 \usepackage[utf8]{inputenc}
4 \usepackage{amsmath}
5 \usepackage{amsfonts}
6 \usepackage{amssymb}
7 \begin{document}
8 \newcommand{\h}[1]{#1}
9 %\renewcommand{\h}[1]
10
11 \title{ LLgramatical }
12 \maketitle
```

Figura E.12: Obtención del cuestionario completo

- Obtención del cuestionario en blanco. Para ello, bastaría con eliminar el símbolo % de delante de la siguiente línea:

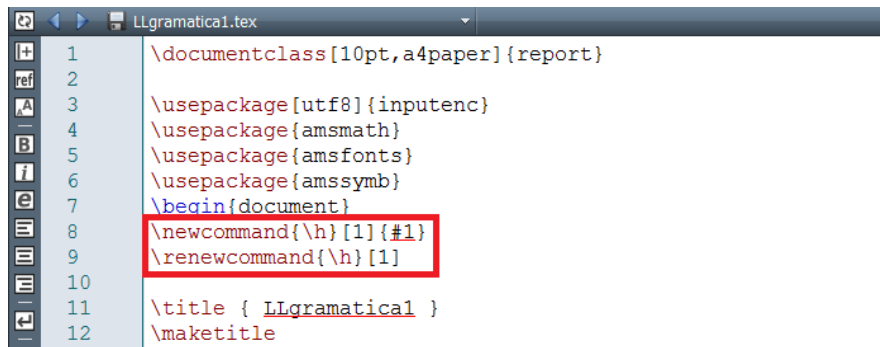
```
%\renewcommand{\h}[1]
```



```
1 \documentclass[10pt,a4paper]{report}
2
3 \usepackage[utf8]{inputenc}
4 \usepackage{amsmath}
5 \usepackage{amsfonts}
6 \usepackage{amssymb}
7 \begin{document}
8 \newcommand{\h}[1]{#1}
9 %\renewcommand{\h}[1]
10
11 \title{ LLgramatical }
12 \maketitle
```

Figura E.13: Obtención del cuestionario en blanco: Descomentar

Así, ambas líneas deberían estar activas.



```
1 \documentclass[10pt,a4paper]{report}
2
3 \usepackage[utf8]{inputenc}
4 \usepackage{amsmath}
5 \usepackage{amsfonts}
6 \usepackage{amssymb}
7 \begin{document}
8 \newcommand{\h}[1]{#1}
9 \renewcommand{\h}[1]
10
11 \title{ LLgramatica1 }
12 \maketitle
```

Figura E.14: Obtención del cuestionario en blanco

Siempre que se compile código, se crea un archivo .pdf en el mismo directorio donde se encuentra el fichero .TEX, con los cuestionarios elegidos.